Ubiquitous Systems and Petri Nets

David de Frutos Escrig*, Olga Marroquín Alonso**,
and Fernando Rosa Velardo*

Abstract. Several years before the popularization of the Internet, Mark Weiser proposed the concept of ubiquitous computing with the purpose of enhancing the use of computers by making many computers available throughout the physical environment, but making them effectively invisible to the user. Nowadays, such idea affects all areas of computing science, including both hardware and software. In this paper, a formal model for ubiquitous systems based on Petri nets is introduced and motivated with examples and applications. This simple model allows the definition of two-level ubiquitous systems, composed of a collection of processor nets providing services, and a collection of process nets requesting those services. The modeled systems abstract from middleware details, such as service discovery protocols, and security infrastructures, such as PKI's or trust policies, but not from mobility or component compatibility.

1 Introduction

The term *ubiquitous computing* was coined by Mark Weiser [11, 12] in order to describe environments full of devices that compute and communicate with its surrounding context and, furthermore, interact with it in a highly distributed but pervasive way. By pervasive we mean that users will not be aware of the existence of such environment, much in the same way as they pay little attention to other technologies, already fully integrated in their everyday life. Thus, ubiquitous computing is a vast field that involves not only many areas of computer science, including hardware components, network protocols, and computational methods, but also social sciences.

Since Weiser's vision [11], a great deal has been achieved, mainly because of advances in micro-electronics, that make possible the design of smaller embedded devices. However, the state of the art is probably not as developed as expected. One of the reasons may be the lack of widely accepted formal models, needed at various levels of abstraction, in order to understand "the probably largest *engineered artifact* in human history" (see [7]).

* Work partially supported by the MCYT project MIDAS, TIC2003-01000.
** Work partially supported by the MCYT project MASTER, TIC2003-07848-C02-01.

O. Gervasi et al. (Eds.): ICCSA 2005, LNCS 3481, pp. 1156–1166, 2005.

Nevertheless, some recently developed formal models can be applied in fields related to ubiquitous computing like workflow, flexible manufacturing or agent-oriented approaches (mobile agents or intelligent agents as in AI research). Among these models we are here interested in those based on Petri nets for their amenable graphical representation and their solid theoretical basis. For instance, the interest of *Elementary Object Systems* [8, 9] has been illustrated in numerous case studies [8]. Elementary Object Systems are composed of a system net and one or more object nets that move along the former, like ordinary tokens of it. Such tokens are able to change their marking, but not their structure, either when lying on a place or when being moved by a transition of the system net. In this way, the change of the object net marking can be either independent from the system net or triggered by it.

In contrast with Elementary Object Systems and their reference semantics, which allow to access a net token from many places at the same time, in *Nested Petri nets* [5, 6] each token is located at a single place at each time. Net tokens may be produced, copied and removed during a system run, as expressed by labels on arcs. The number of those tokens, as well as the level of nestedness, is unlimited, thus obtaining multi-level nested systems, whose behaviour consists of three kinds of steps: An autonomous step in a given level of a Nested Petri net follows the ordinary firing rule for high-level Petri nets; horizontal synchronization is defined as the simultaneous firing of two element nets located in the same place of a system net; and vertical synchronization is the firing of a system net together with the firing of its token nets that are involved.

In earlier papers [3, 4], we have introduced another multi-level extension of the Elementary Object Systems called *Ambient Petri nets*, which allows the arbitrary nesting of ambients permitted in the Ambient Calculus [1]. As a consequence, it is possible to find in the places of an Ambient Petri net both ordinary and high-level tokens. The latter move along the net due to the firing of ambient transitions, labeled by capabilities that are obtained from names: Given a name n of a component net, that is, a bounded place where computation happens, the capability $in\, n$ allows to enter into n, the capability $out\, n$ allows to exit out of n, and the capability $open\, n$ allows to open n. Besides, ordinary transitions consume and produce only ordinary tokens by following the firing rule from ordinary Petri nets. In [4] the basic model of Ambient Petri nets has been extended with the aim of supporting the replication operator from the Ambient Calculus, $!P$, which generates an unbounded number of parallel replicas of P. By combining these elements, together with concepts such as limitation of access to locations, Ambient Petri nets provide a framework to describe wide area network mobility.

Although the described models were not originally conceived in the framework of ubiquitous computing, they can be used for handling some of its most important aspects, specially mobility. Nevertheless, many features of ubiquitous computing, such as the supply and demand of resources between processors and processes, context awareness, and ad-hoc nets, are not naturally modeled. In order to formalize such features, in this paper we define a new model based on

Petri nets called *Ubiquitous nets*, whose basic version relies on ordinary Petri nets, though it can be easily enhanced by considering coloured Petri nets.

Ubiquitous nets allow to model both devices (processors) and software components located in processors (processes), in such a way that processes change their location due to the firing of special *movement transitions*. Besides, we abstract from middleware details, such as those dealing with service discovery or transport protocols, so transitions offering or requesting a service are detected by others just by its mere existence. Then, a service is supplied whenever its offer and request are co-located, that is, whenever the firing of the corresponding *service-supply* and *service-request* transitions can be done at the same time. The synchronization criteria is merely syntactical, that is, two transitions can synchronize whenever the corresponding labels match. Nevertheless, in real open systems we could always make use of specific-domain ontologies, in order to avoid this lack of flexibility.

The paper is structured as follows. Section 2 gives the formal definition of ubiquitous nets by considering the simplest possible model in which processes move from location to location with no processor interaction. Section 3 illustrates those definitions with a simple example composed of three processor nets and a process net that is required to follow an authentication protocol in order to obtain a specific service. Extensions of the basic model are introduced and motivated in Section 4. Finally, conclusions and areas for further study are discussed in Section 5.

2 Formal Definitions

Ubiquitous systems are defined in order to model the supply and demand of services/resources of both processors and processes, respectively. To define the exchange of such services, we consider a countable alphabet of labels $\mathcal{S}$, which will denote available resources. Moreover, we assume the existence of two bijections $!:\mathcal{S} \to \mathcal{S}^!$ and $?:\mathcal{S} \to \mathcal{S}^?$, by means of which we associate to each label $s \in \mathcal{S}$ two synchronizing actions, $s! \in \mathcal{S}^!$ and $s? \in \mathcal{S}^?$, respectively. The countable alphabet of labels $\mathcal{S}^!$ will denote resources provided by processor nets, while the countable alphabet of labels $\mathcal{S}^?$ will denote resources requested by process nets. Besides, we consider a countable alphabet of labels $\mathcal{A}$, which will denote autonomous actions performed by either processors or processes.

In order to identify the different components of a ubiquitous system, we start with two given sets of processor names, $\mathcal{N}_r$, and process names, $\mathcal{N}_s$, thus taking $\mathcal{N} = \mathcal{N}_r \cup \mathcal{N}_s$. Then we have:

Definition 1. *A* ***processor net*** *is a labeled Petri net* $L = (P, T, F, \lambda)$ *where:*

- P *and* T *are disjoint sets of places and transitions.*
- $F \subseteq (P \times T) \cup (T \times P)$ *is the set of arcs of the net.*
- λ *is a function from* T *to the set* $\mathcal{A} \cup \mathcal{S}^!$.

Definition 2. *A **process net** is a labeled Petri net $A=(P,T,F,\lambda)$ where:*

- *P and T are disjoint sets of places and transitions.*
- *$F\subseteq(P\times T)\cup(T\times P)$ is the set of arcs of the net.*
- *λ is a function from T to the set $\mathcal{A}\cup\mathcal{S}^{?}\cup\mathcal{M}_s$, where $\mathcal{M}_s=\{go_l \mid l\in\mathcal{N}_r\}$.*

As stated above, ubiquitous systems are composed of a collection of processor nets which provide services to a collection of process nets. Therefore, a processor net has two types of transitions: *autonomous transitions* (those with $\lambda(t)\in\mathcal{A}$), and *service-supply transitions* (those with $\lambda(t)\in\mathcal{S}^{!}$). Those services are requested by process nets, which migrate from processor to processor due to the execution of *go* transitions. Therefore, a process net has three types of transitions: *autonomous transitions* (those with $\lambda(t)\in\mathcal{A}$), *service-request transitions* (those with $\lambda(t)\in\mathcal{S}^{?}$), and *movement transitions* (those with $\lambda(t)\in\mathcal{M}_s$).

Definition 3. *A **plain ubiquitous system** is a pair of the form $\mathbb{U}=\langle\mathbb{R},\mathbb{S}\rangle$ where $\mathbb{R}=\{l^1:L^1,\ldots,l^m:L^m\}$ is a finite collection of named processor nets or locations and $\mathbb{S}=\{a^1:A^1,\ldots,a^n:A^n\}$ is a finite collection of named process nets or agents, with $m>0$, $n\geq 0$, $\forall k\in\{1,\ldots,m\}$ $l^k\in\mathcal{N}_r$ and $L^k=(P_l^k,T_l^k,F_l^k,\lambda_l^k)$, and $\forall k\in\{1,\ldots,n\}$ $a^k\in\mathcal{N}_s$ and $A^k=(P_a^k,T_a^k,F_a^k,\lambda_a^k)$.*

Note that, in contrast with Ambient Petri nets, ubiquitous nets define two-level systems with no distinguished root net, so we consider that all processors and process nets have the same significance. The static nature of processor nets, whose location is fixed, highlights the static character of some devices such as operating systems. On the other hand, the dynamic nature of process nets, that move from processor to processor in order to request the execution of services, reflects the behaviour of processes performed in a distributed way by the whole distributed system.

The current location of processes is described by means of a location function *loc*, which, given a process net, returns its communicating processor. In the following, we assume that in an ubiquitous system each component net has a different name. $\mathcal{N}_r(\mathbb{U})=\{l^1,\ldots,l^m\}$ will denote the set of processor names in $\mathbb{U}$, and $\mathcal{N}_s(\mathbb{U})=\{a^1,\ldots,a^n\}$ will denote the set of process names in $\mathbb{U}$.

Definition 4. *Given a plain ubiquitous system $\mathbb{U}$, we define a **location function** for $\mathbb{U}$ as a function $loc:\mathcal{N}_s(\mathbb{U})\to\mathcal{N}_r(\mathbb{U})$. A **located ubiquitous system** is a plain ubiquitous system for which we have defined a location function.*

A located ubiquitous system describes the structure of both processors and processes. In order to suitably represent their state, we use the usual concept of marking, in such a way that places are occupied by ordinary tokens that move along the system by following the ordinary firing rule.

Definition 5. *A **dynamic located ubiquitous system** is a located ubiquitous system for which we have defined an ordinary marking $M:P\to\mathbb{N}$, where P is the full set of places of the ubiquitous system, that is, $P=(\bigcup_{k=1}^{m}P_l^k)\cup(\bigcup_{k=1}^{n}P_a^k)$.*

Similarly, we define the full sets of transitions and arcs of the ubiquitous system as the sets $T=(\bigcup_{k=1}^{m} T_l^k)\cup(\bigcup_{k=1}^{n} T_a^k)$ and $F=(\bigcup_{k=1}^{m} F_l^k)\cup(\bigcup_{k=1}^{n} F_a^k)$, respectively, and the full labelling function of the system as $\lambda(t)=\lambda_c^k(t)$ if $t\in T_c^k$ with $c\in\{a,l\}$.

As stated before, both processors and processes can perform *autonomous transitions* that model independent actions, that is, actions whose execution does not depend on the surrounding context of the evolving net. The corresponding firing rule is the one for ordinary Petri nets, since the location function does not change. Besides, a process can move from its current location to any other due to the execution of *movement transitions*, labeled by go_l with $l\in\mathcal{N}_r$, which specify the new destination l. Note that in this basic model we disregard security issues, so we consider that the full set of processor names is known to the full collection of processes of the system. Finally, the supply of a service s is modelled by means of the synchronized firing of two transitions, $t_1\in T_a^k$ and $t_2\in T_l^{k'}$ with $\lambda_a^k(t_1)=s?$ and $\lambda_l^{k'}(t_2)=s!$, corresponding respectively to the request of the service by the process net A^k, and its offering by the processor net $L^{k'}$. Those transitions can only be fired simultaneously, each of them following the firing rule for ordinary Petri nets.

Definition 6. *Let $\langle\mathbb{U},loc\rangle$ be a located ubiquitous system and M be a marking of it. An **autonomous** transition $t\in T$, with $\lambda(t)\in\mathcal{A}$, is enabled at marking M if $\forall p\in{}^\bullet t\ M(p)>0$. The reachable state of $\mathbb{U}$ after the firing of t is that described as follows:*

- *The reachable marking M' is defined by $M'(p)=M(p)-F(p,t)+F(t,p)$ for all $p\in P$.*
- *The location function loc does not change.*

Definition 7. *Let $\langle\mathbb{U},loc\rangle$ be a located ubiquitous system and M be a marking of it. A **movement** transition $t\in T_a^k$, with $\lambda_a^k(t)=go_l\in\mathcal{M}_s$, is enabled at marking M if $\forall p\in{}^\bullet t\ \ M(p)>0$. The reachable state of $\mathbb{U}$ after the firing of t is that described by:*

- *The reachable marking M' is defined by $M'(p)=M(p)-F(p,t)+F(t,p)$ for all $p\in P$.*
- *The current location of process net a^k changes, getting $loc(a^k)=l$. The location of the rest of the process nets remains the same.*

Definition 8. *Let $\langle\mathbb{U},loc\rangle$ be a located ubiquitous system and M be a marking of it. A pair of **service-supply/service-request** transitions (t_1,t_2), with $t_1\in T_a^k$ and $t_2\in T_l^{k'}$, $\lambda(t_1)=s?\in\mathcal{S}^?$ and $\lambda(t_2)=s!\in\mathcal{S}^!$, and $loc(a^k)=l^{k'}$, is enabled at marking M if $\forall p\in{}^\bullet t_1\cup{}^\bullet t_2\ \ M(p)>0$. The reachable state of $\mathbb{U}$ after the firing of (t_1,t_2) is that described as follows:*

- *The reachable marking M' is defined by*

$$M'(p)=M(p)-\sum_{v\in\{t_1,t_2\}}F(p,v)+\sum_{v\in\{t_1,t_2\}}F(v,p)\quad\forall p\in P$$

- *The location function loc does not change.*

Due to the firing rule in Definition 8, services are provided in mutual exclusion with the purpose of avoiding their concurrent use, since they are considered unshareable resources. In this way, whenever a process requests a service s to its processor, both nets must synchronize the firing of the corresponding transitions in order to satisfy such demand. At that moment, service s becomes unavailable for any process requesting the same resource. Note that if there exist more than one process net demanding such service, the choice among them is made in a non-deterministic way.

3 A Simple Application

In order to illustrate the behaviour of ubiquitous nets, in this section we present an example that models a system composed of three processor nets, L^1, L^2 and L^3, and a process net, A, initially located in L^3 (Figure 1). Processor L^1 can be interpreted as an electronic notes system [2] that requires authentication to view its contents (action identified as service $s2$), and L^2 can be seen as an electronic thermometer [10] in which the action of consulting the temperature is denoted by $s3$. Both processors can also give the local time, which is denoted as service

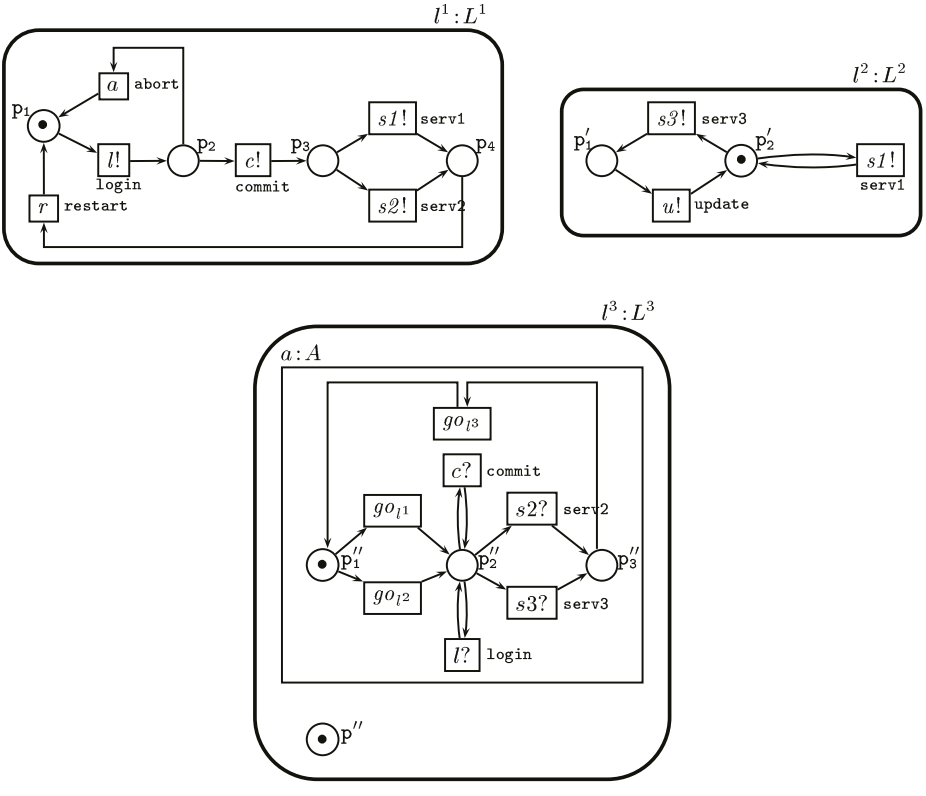

Fig. 1. An ubiquitous system modeled by Petri nets

$s1$. Processor net L^3, composed of a single place and no transitions, can neither evolve in an independent way nor interact with any process, since it is just a container that allows to store agent nets.

Therefore, process A can move either to L^1 or L^2, where it demands services $s2$ or $s3$ (trying to view the contents of the notes system or asking for the temperature, respectively). In order to supply service $s2$, processor L^1 requires first to log in and if it is successfully done then to proceed with a commitment (otherwise it aborts), whereas processor L^2 does not demand any such authentication to offer its services. On the other hand, after the firing of a movement transition, process A may not only log in if asked, but also try to commit with no previous logging, thus trying to force the authentication protocol.

In this scenario, it is clear that after the firing of transition go_{l^2}, process A obtains service $s3$ from its new location L^2, which must be updated before offering again its services (for instance, checking the temperature again). Then, process A needs to restart before going back to its initial state, where now it may choose to execute transition go_{l^1}. As a consequence of this firing, process A moves to L^1, that demands it to log in before proceeding with a commitment, which is needed to supply service $s2$.

4 Extensions of the Basic Model

Ubiquitous nets allow to define two-level systems focusing on both the supply and demand of services and the mobility of processes. Nevertheless, their simplicity produces some drawbacks, that can be easily removed by introducing some extensions in the defined basic model.

In the first place, real systems constrain both mobility of processes and access to their resources as a general rule. In particular, processes do not migrate by themselves, but are moved by processors. As a consequence, it is reasonable to limit the access to processors depending on their current availability to receive processes. Moreover, processes would need to obtain the permission of their present location to move away to the desired processor.

Therefore, in general it is necessary to model a three-way synchronization among the moved process, its current location and its new destination. In order to do it we introduce processor transitions labelled by lgo_l and lin. Their intended meaning is that the processor firing a transition labelled by lgo_l allows any process that can fire transition go_l to exit out of it. This migration can only take place when the destination processor, l, executes the admission transition lin at the same time.

Definition 9. *A **go-processor net** is a Petri net $L_{go} = (P, T, F, \lambda)$ where:*

- *P and T are disjoint sets of places and transitions.*
- *$F \subseteq (P \times T) \cup (T \times P)$ is the set of arcs of the net.*
- *λ is a function from T to $\mathcal{A} \cup \mathcal{S}^! \cup \mathcal{M}_r$, where $\mathcal{M}_r = \{lin\} \cup \{lgo_l \mid l \in \mathcal{N}_r\}$.*

Definition 10. *Let $\langle \mathbb{U}, loc \rangle$ be a located ubiquitous system with go-processor nets and M be a marking of it. A tuple of **movement** transitions (t_1, t_2, t_3), with*

$t_1 \in T_l^i$, $t_2 \in T_a^k$ and $t_3 \in T_l^{i'}$, $\lambda(t_1) = lgo_{l^{i'}}$, $\lambda(t_2) = go_{l^{i'}}$ and $\lambda(t_3) = lin$, and $loc(a^k) = l^i$, is enabled at marking M if $\forall p \in {}^\bullet t_1 \cup {}^\bullet t_2 \cup {}^\bullet t_3$ $M(p) > 0$. The reached state of $\mathbb{U}$ after the firing of (t_1, t_2, t_3) is that described by:

- *The reachable marking M' is defined by*

$$M'(p) = M(p) - \sum_{v \in \{t_1, t_2, t_3\}} F(p, v) + \sum_{v \in \{t_1, t_2, t_3\}} F(v, p) \quad \forall p \in P$$

- *The current location of the process net a^k changes, getting $loc(a^k) = l^{i'}$. The location of the rest of the process nets remains the same.*

In this extended model, authentication could be simply modeled: We only have to replace *lin* transitions by others labeled by lin_a, where a is the name of the incoming process, that is, the one the destination location is ready to receive. In this way, it is easy to limit the access to some services by taking into account the names of processes, and hence a processor net will only admit those agents whose name appears in labels of the form lin_a.

However, in this model processes must include in their code concrete information about their desired movements, although in some cases such movements are performed in a non-deterministic way (this happens, for instance, whenever there exists a choice between movement transitions). Furthermore, processors must have a static knowledge of the names of those processes whose entry is allowed. Nevertheless, this assumption is a bit coarse, and more flexible mechanisms to control authentication and mobility are desirable. Regarding the latter, processor names will appear as token values. Then, a synchronized firing of the movement transitions, that in this new version of the model would be labeled by *lgo*, *go* and *lin*, respectively, produces the migration of the involved process to the location indicated by the consumed token. In this way, labels of tokens represent a permission or a capability to enter into the corresponding processors.

Definition 11. *A **ggo-processor net** is a Petri net $L_{ggo} = (P, T, F, \lambda)$ where:*

- *P and T are disjoint sets of places and transitions.*
- *$F \subseteq (P \times T) \cup (T \times P)$ is the set of arcs of the net.*
- *λ is a function from T to the set $\mathcal{A} \cup \mathcal{S}^! \cup \{lin, lgo\}$.*

Definition 12. *A **ggo-process net** is a Petri net $A_{ggo} = (P, T, F, \lambda)$ where:*

- *P and T are disjoint sets of places and transitions.*
- *$F \subseteq (P \times T) \cup (T \times P)$ is the set of arcs of the net.*
- *λ is a function from T to the set $\mathcal{A} \cup \mathcal{S}^? \cup \{go\}$.*
- *Each $t \in T$ such that $\lambda(t) = go$ has a distinguished precondition, $where_t \in P$, whose tokens should be labelled with processor names.*

In order to define the behaviour of ggo-systems, we have to separate ordinary places from those distinguished ones storing processor names. This could be easily formalized using a simple version of coloured Petri nets, though its definition could be rather cumbersome. Here we propose a less formal but clearer definition.

Definition 13. *A **dynamic located ubiquitous ggo-system** is a located ubiquitous system with ggo-processor and ggo-process nets, for which we have defined a marking $M : P \to \mathbb{N} \cup \mathcal{M}(\mathcal{N}_r(\mathbb{U}))$, where P is the set of places of the ubiquitous system, and all the tokens are ordinary ones except from those in distinguished places, that is, $M(p) \in \mathbb{N}\ \forall p \in P \setminus P_{go}$ and $M(p) \subseteq \mathcal{M}(\mathcal{N}_r(\mathbb{U}))\ \forall p \in P_{go}$, where P_{go} is the set of distinguished places of the ubiquitous system $P_{go} = \{where_t \mid t \in T, \lambda(t) = go\}$.*

Therefore, each process in a ggo-system has some distinguished places in P_{go} connected as preconditions of its *go* transitions. In this way, whenever a three-way synchronization is performed, the name of the new location is taken from the distinguished place $where_t$, taking as t the corresponding transition labeled by *go*. Then, it is simple to model a mechanism for the transmission of processor names to processes by means of special agents that move from location to location providing the corresponding process nets with their labeled tokens. Such mechanism is not formalized here due to lack of space.

Definition 14. *Let $\langle \mathbb{U}, loc \rangle$ be a located ubiquitous ggo-system and M be a marking of it. A tuple of **movement transitions** (t_1, t_2, t_3), with $t_1 \in T_l^i$, $t_2 \in T_a^k$ and $t_3 \in T_l^{i'}$, $\lambda(t_1) = lgo$, $\lambda(t_2) = go$ and $\lambda(t_3) = lin$, $loc(a^k) = l^i$ and $l^{i'}$ is a processor name stored in the distinguished precondition $where_{t_2}$ of t_2, is enabled at marking M if $\forall p \in {}^\bullet t_1 \cup {}^\bullet t_2 \cup {}^\bullet t_3\ \ M(p) > 0$. The reachable state of $\mathbb{U}$ after the firing of (t_1, t_2, t_3) is that described as follows:*

- *The reachable marking M' is defined by*

$$M'(p) = M(p) - \sum_{v \in \{t_1,t_2,t_3\}} F(p,v) + \sum_{v \in \{t_1,t_2,t_3\}} F(v,p) \quad \forall p \in P \setminus P_{go}$$
$$M'(p) = M(p) \quad \forall p \in P_{go}$$

- *The current location of the process net a^k changes, getting $loc(a^k) = l^{i'}$. The location of the rest of the process nets remains the same.*

Following the above definition, the marking of places in P_{go} does not change due to the firing of transitions, since the distinguished input place of a *go* transition is just a container for the names of the available destinations. The coloured formal version of ggo-systems would be more flexible, allowing us to indicate how the tokens annotated with processor names are transmitted and consumed, in such a way that processes can have a dynamic knowledge of their possible destinations.

5 Conclusions and Future Work

We have introduced a model for two-level ubiquitous systems, in which a collection of processors provide services to a collection of processes that request those services. In the simplest version of the model, processors remain fixed in their locations, whereas processes move from processor to processor in order to obtain the resources they need.

Supply and demand of services is modeled by the synchronized firing of two transitions: a *service-demand transition* located in the involved process and a *service-supply transition* located in the corresponding processor. Besides, mobility is formalized by the execution of *movement transitions*, by means of which each process sets its destinations.

This simple model is then enhanced with some extensions that include, first a three-way synchronization mechanism that limits resource access (a process moves if and only if its current processor lets it go and its new location lets it in), and then introduces the colouring of some tokens, that allow to dynamically determine the destinations of the moving processes.

As work in progress, we are currently introducing new features in our model to cover the most of the characteristic properties of ubiquitous computing, mainly a procedure to dynamically transmit private processor names to processes, in such a way that access to locations can be adequately constrained. We will do that by using a simple version of coloured Petri nets. Certainly, this will lead us to the analysis of security properties by means of, for example, typing mechanisms to suitably restricting the contents of net places. In addition to this, we will generalize the described framework in order to encompass the dynamic generation of new processes during a system run. With this purpose, we will introduce a set of process types $\{A_1, A_2, \ldots, A_k\}$, which are ordinary process nets initialized in such a way that the firing of a special transition $create_i$ will generate a new copy of the process of type i.

References

1. L. Cardelli. *Mobility and Security.* Proceedings of the NATO Advanced Study Institute on Foundations of Secure Computation, pp.3-37. IOS Press, 2000.
2. K. Cheverst, A. Dix, D. Fitton and M. Rouncefield. *'Out To Lunch': Exploring the Sharing of Personal Context through Office Door Displays.* Proceedings of the Australasian Computer-Human Conference-OzCHI 2003, pp.74-83. 2003.
3. D. Frutos Escrig and O. Marroquín Alonso. *Ambient Petri Nets.* Foundations of Global Computing 2003, ENTCS vol.85, 27 pp. Elsevier Science, 2003.
4. D. Frutos Escrig and O. Marroquín Alonso. *Replicated Ambient Petri Nets.* Computational Science-ICCS 2003, LNCS vol.2658, pp.774-783. Springer-Verlag, 2003.
5. I.A. Lomazova. *Nested Petri Nets; Multi-level and Recursive Systems.* Fundamenta Informaticae vol.47, pp.283-293. IOS Press, 2002.
6. I.A. Lomazova. *Modeling Dynamic Objects in Distributed Systems with Nested Petri Nets.* Fundamenta Informaticae vol.51, pp.121-133. IOS Press, 2002.

7. R. Milner. *Theories for the Global Ubiquitous Computer*. Foundations of Software Science and Computation Structures-FoSSaCS 2004, LNCS vol.2987, pp.5-11. Springer-Verlag, 2004.
8. R. Valk. *Petri Nets as Token Objects: An Introduction to Elementary Object Nets*. Applications and Theory of Petri Nets 1998, LNCS vol.1420, pp.1-25. Springer-Verlag, 1998.
9. R. Valk. *Concurrency in Communicating Object Petri Nets*. Concurrent Object-Oriented Programming and Petri Nets, LNCS vol.2001, pp.164-195. Springer-Verlag, 2001.
10. R. Want. *Enabling Ubiquitous Sensing with RFID*. Computer vol.37(4), pp.84-86. IEEE Computer Society Press, 2004.
11. M. Weiser. *Some Computer Science Issues in Ubiquitous Computing*. Communications of the ACM vol.36(7), pp.74-84. ACM Press, 1993.
12. M. Weiser. *The Computer for the 21st Century*. Proceedings of Human-computer Interaction: Toward the Year 2000, pp.933-940. Morgan Kaufmann Publishers Inc, 1995.